

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-330233

(43) 公開日 平成9年(1997)12月22日

(51) Int.Cl. <sup>8</sup>	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 9/45			G 0 6 F 9/44	3 2 2 F
11/28	3 3 0		11/28	3 3 0 A
			9/44	3 2 2 M

審査請求 未請求 請求項の数 1 O L (全 15 頁)

(21) 出願番号 特願平9-55902

(22) 出願日 平成9年(1997)3月11日

(31) 優先権主張番号 6 1 6, 6 0 8

(32) 優先日 1996年3月15日

(33) 優先権主張国 米国 (U S)

(71) 出願人 590000400

ヒューレット・パカード・カンパニー

アメリカ合衆国カリフォルニア州パロアル

ト ハノーバー・ストリート 3000

(72) 発明者 ウィリアム・ビー・バズビー

アメリカ合衆国94019カリフォルニア州ハ

ーフ・ムーン・ベイ、カサ・デル・マー・

ドライブ 404

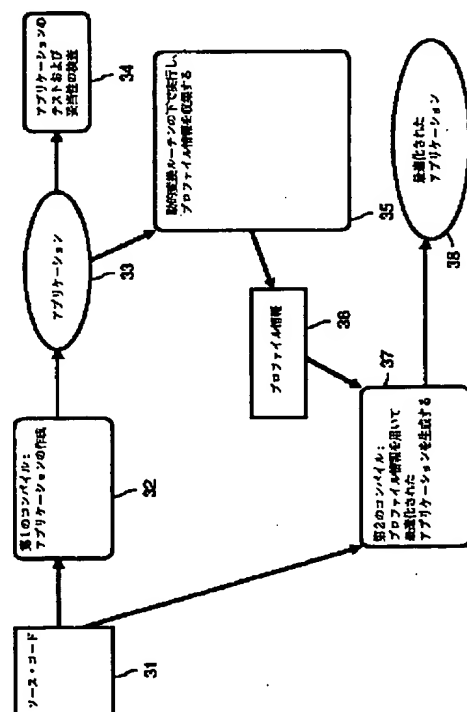
(74) 代理人 弁理士 岡田 次生

(54) 【発明の名称】 最適目的コード生成方法

(57) 【要約】

【課題】 プロファイルに基づいて目的コードを最適化する場合のコンパイル回数を削減し、コンパイルの効率化を図る。

【解決手段】 アプリケーションのソース・コードをコンパイルして、第1の目的コードを生成する。アプリケーション最適化のために必要となるプロファイル情報を生成する機能を持つプロファイル作成コードを含むように、本発明の動的変換ルーチンによって第1の目的コードを第2の目的コードに動的に変換する。次に、第2の目的コードを実行して、上記プロファイル作成コードに基づいて、プロファイル情報を生成する。最後に、このプロファイル情報を使用してアプリケーションのソース・コードを再コンパイルし、最適化された目的コードを生成する。



## 【特許請求の範囲】

【請求項1】アプリケーションに関する最適化された目的コードを生成する方法であって、アプリケーションのソース・コードをコンパイルして該アプリケーションに関する第1の目的コードを生成するステップと、実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを含む第2の目的コードに上記第1の目的コードを変換し、該第2の目的コードを実行してプロファイル情報を生成することを含む上記第1の目的コードに関するプロファイル情報を生成するステップと、上記プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成するステップと、を含む最適目的コード生成方法。

## 【発明の詳細な説明】

## 【0001】

【発明の属する技術分野】本発明は、コンパイラ最適化に関するもので、特に最適化コンパイル・システムにおいて実行時情報を収集し活用するための動的変換の使用に関するものである。

## 【0002】

【従来の技術】プログラムは、一般に高水準プログラム言語で書かれる。しばしばソース・コードと呼ばれるこの高水準言語は、コンパイラ・プログラムによってアセンブリ言語に変換される。目的コードと呼ばれるバイナリ形式のアセンブリ言語は、コンピュータによって実際に実行されるコード形式である。目的コードは、先ず、リンカによって相互にリンクされる目的コード・モジュールの形式で生成される。本明細書において、用語「コンパイル」は、目的コード・モジュールを生成するプロセスおよび目的コード・モジュールを相互にリンクするプロセスの両者を含む。

【0003】目標コンピュータによって実行される目的コードの処理性能を向上させるため、コンパイルされたコードはしばしば最適化される。目的コードはいくつかの異なる形態で最適化される。

【0004】最適化の1つのタイプは、プロファイルに基づく最適化(profile-based optimizationの頭文字をとってPBOと呼ばれる)である。PBOにおいては、コンパイラは、典型的な入力データセットを用い、アプリケーションの実行を通して得られるプロファイル・データに基づいてアプリケーションを最適化する。例えば、特定の複数のプロシージャが頻繁に互いを呼び出すとすれば、リンカは、目的コード・ファイルの中でそれらプロシージャを互いに近い位置に置き、それにより、アプリケーション実行中の命令キャッシュ・ミス、変換ルックアサイド・バッファ(TLB)ミス、および記憶ページ・フォールトを減少させることができる。同様の最

(2)

適化は、プロシージャの基本ブロックのレベルでも行うことができる。基本ブロックまたは分節は、開始点および終了点以外には分岐が存在しないようなアセンブリ・コードの1つの連続的節である。プロファイル・データは、また、コード・スケジューリングおよびレジスタ割り付けのような他の一般的タスクのためにもコンパイラによって使用される。

【0005】PBOを実行するためには、アプリケーションのソース・コードは、プロファイル作成用の特別コードを挿入することによって目的コードに仕組みを組み込む特別なコンパイラ・オプションを使用してコンパイルされる。アプリケーションが実行されると、その特別プロファイル作成コードによってプロファイル・データが作成される。次にプロファイル・データが収集される。アプリケーションのソース・コードが再コンパイルされる時、プロファイル・データを使用して、アプリケーション・ソース・コードの新しいコンパイルが一層最適化される。PBOの詳細は、例えば、ヒューレット・パッカード社発行の“Programming on HP-UX, HP 9000 Series 700/800 Computers”(HP Part No. B2355-90652, January 1995, pp. 8-1 through 8-17)に記載されている。

## 【0006】

【発明が解決しようとする課題】PBOの使用にかかわる1つの問題は、PBOのための別のコンパイルの実行が必要とされる点である。従って、ユーザがPBOを利用することを望む場合、典型的にはアプリケーション・ソース・コードは3回コンパイルされることが必要である。第1回は、アプリケーションをテストし妥当性を検査するために使用されるアプリケーションのコピーを作成するためのものである。コードの第2回のコンパイルでは、プロファイル・データを作成するためプロファイル作成用特別コードが追加される。次に、コンパイルされたアプリケーションを最適化するため、生成されたプロファイル・データを使用してアプリケーションを構築する第3のコンパイルが必要である。このように独立したコンパイルを行うことなくPBOを実施できる方法が求められている。

## 【0007】

【課題を解決するための手段】本発明に従う動的変換を使用して、アプリケーションの目的コードを最適化するために使用されるプロファイル情報が生成される。アプリケーションに関する最適化された目的コードを生成するため、アプリケーションのソース・コードがコンパイルされ、アプリケーションに関する第1の目的コードが生成される。第1の目的コードは、アプリケーションをテストして妥当性を検査するために使用することもできる。第1の目的コードは、また、アプリケーションに関するプロファイル情報の生成に使用することもできる。これは、第1の目的コードを第2の目的コードに動的に

変換することによって実施される。この変換の際に、実行時にプロファイル情報を生成するプロファイル作成コードが第2の目的コードに、含まれる。生成された第2の目的コードは次にプロファイル情報を生成するため実行される。最後に、アプリケーションのソース・コードは、最適化された目的コードを作成するため、生成したプロファイル情報を活用して再コンパイルされる。

【0008】本発明は、具体的には、アプリケーションに関する最適化された目的コードを生成するため、アプリケーションのソース・コードをコンパイルして該アプリケーションに関する第1の目的コードを生成するステップ、実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを含む第2の目的コードに上記第1の目的コードを変換し、該第2の目的コードを実行してプロファイル情報を生成することを含む上記第1の目的コードに関するプロファイル情報を生成するステップ、および上記プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成するステップを含む最適化された目的コード生成方法を提供する。

【0009】アプリケーションの目的コードを更に最適化するため最適化された目的コードに関する追加のプロファイル情報を生成するため、このプロセスを繰り返すこともできる。

【0010】更に、本発明の別の局面では、上記第1の目的コードに注釈が配置される。コンパイラの変換ルーチンが第1の目的コード内の注釈を利用して、第2の目的コード内に置かれるべき特定のプロファイル作成コードに従って生成されるプロファイル情報を決定する。注釈は、コンパイラによって第1の目的コードに置かれる。代替的にあるいは追加として、目的コード内の注釈および目的コードの実行の結果生成されるプロファイル情報に対するアクセス機能をユーザに与えることができる。ユーザは最適化プロセスを指示するため目的コード内に注釈を追加または変更することができる。このプロセスは、それ以上の最適化が見いだされなくなるまで、または最適化されたアプリケーションの処理性能がユーザにとって満足できるものとなるまで、反復プロセスとして実施することができる。

#### 【0011】

【発明の実施の形態】図1は、動的変換ルーチン15を使用してアプリケーション10を実行するコンピュータ・システム7のブロック図である。コンパイラが実行の前に実行可能コードを生成するのに対し、動的変換ルーチン15は実行時に変換されたコード8を生成する点がコンパイラと相違する。アプリケーション10のコード・ブロックが実行時に変換される。変換されたコード・ブロックは、図1において変換されたコード8として示されている。変換されたコード8は、何度も実行されるコード・ブロックの各々が実行時に一度だけ変換され

るだけでよいように、キャッシュ・メモリのようなメモリに記憶される。このアプローチは、実行前にコードが変換されることを必要としない柔軟性を実現するだけでなく、実行毎にコード・ブロックを変換しなければならないとした場合に生じるオーバーヘッドをも減少させる。

【0012】本発明の好ましい実施形態において、変換されたコード8は変換コード・キャッシュ20に記憶される。変換コード・キャッシュ20が一杯の時、新しく変換されるコード・ブロックのために空間を作成するため以前に変換されたコード・ブロックの一部が破棄される必要があるかもしれない。これは、破棄されたコード・ブロックが再使用されるならば再変換されるべきことを必要とするが、メモリ使用における潜在的節約を可能にする。以前に変換されたコード・ブロックを破棄する代わりに、それらをシステム・メモリに記憶することもできる。

【0013】図2は、本発明の好ましい実施形態に従ってプロファイル・データを生成する動的変換ルーチンを利用するシステムの流れ図である。ソース・コード31は、ステップ32において、アプリケーション33を作成するためコンパイルされる。コンパイルされたアプリケーション33は、ステップ34で、テストされ妥当性を検査されることもある。アプリケーション33は、テストおよび妥当性検査の間全速力で実行できる。コンパイル時にアプリケーション33にプロファイル作成コードが加えられなかったため、アプリケーション33の処理性能は、アプリケーション33内へのプロファイル作成コードの内包によって妨げらることがない。

【0014】ステップ35において、プロファイル情報を収集するためアプリケーション33は動的変換ルーチンの下で実行される。変換ルーチンは目的コードから目的コードへの変換を実行する。すなわち、アプリケーション33の目的コードは、変換ルーチンによって変換された目的コードに変換される。次に変換された目的コードが実行される。ステップ35で生成される変換された目的コードとステップ32でアプリケーション33を作成するためコンパイルされた目的コードの間の相違は、プロファイル作成コードがステップ35で作成された変換目的コードに加えられる点である。換言すれば、ステップ35で生成された変換目的コードが、実行時にプロファイル情報36を生成するために使用されるプロファイル作成コードを付加的に含む点を除いて、ステップ35で生成された変換目的コードはステップ32でアプリケーション33を作成するためコンパイルされた目的コードと同等または本質的に同じである。

【0015】ステップ37において、プロファイル情報36を使用して第2のコンパイルの間最適アプリケーション38が作成される。要するに、プロファイルに基づく最適化(PBO)処理の実行のためアプリケーション・

ソース・コードは2回だけコンパイルされる。第1回は、アプリケーションをテストし妥当性を検査するために使用されるアプリケーションのソース・コード・コピーを作成するためである。アプリケーションをテストし妥当性を検査するために使用されたものと同じアプリケーション・コピーがプロファイル・データを生成するため変換ルーチンの下で実行される。次に、コンパイルされたアプリケーションを更に最適化するためプロファイル・データを使用してアプリケーションを構築する第2のコンパイルが実行される。

【0016】図3は、アプリケーション10のコードを動的に変換して実行する動的変換および実行プロセス39の使用形態を示している。変換は、プロファイル作成コードを備えた変換目的コードを生成する。アプリケーション10は、命令11およびデータ12を含むように示されている。アプリケーション10内の命令11が実行されるにつれて、ステップ16において、プログラム・カウンタがアプリケーション10内でコード13の新しいブロックをポイントする度毎に、その新たなコード・ブロックが動的変換および実行プロセス39によって既に変換されたものであるか否かが判断される。変換済みでなければ、ステップ18において、そのコード・ブロックはホスト・コンピュータ・システムに対して適切な目的コードに変換される。加えて、プロファイル・データを生成するプロファイル作成コードが追加される。上述のように、変換されるべきコード・ブロックは既にホスト・コンピュータ・システム固有のものとなっているので、変換は、プロファイル作成コードを目的コードに組み入れるためだけに行われる目的コードから目的コードへの変換にすぎない。ステップ17において、コード・ブロック13に関する変換されたコードが実行される。

【0017】図4は、動的変換および実行プロセス39によってアプリケーションを実行するプロセスを更に示している。ステップ21でアプリケーションの実行は始まる。ステップ22において実行されるべき次のアプリケーション・アドレスが調べられる。これは、例えばブロック情報マップへのアドレス25を使用して行われる。ブロック情報マップへのアドレス25はアプリケーションにおけるコード・ブロックのエントリを含み、更に、アプリケーションによって呼び出される共有ライブラリにおけるコード・ブロックのエントリを含む場合もある。本発明の好ましい実施形態においては、実行のため実際に必要とされるコード・ブロックに関するアドレスだけがブロック情報マップ25内に含まれる。これらのアドレスは、コード・ブロックが必要とされる都度、実行時に追加される。

【0018】コード・ブロックは、1つまたは複数の命令からなるグループである。コード・ブロックは、(「基本ブロック」と同様に)分岐によって終了する単一

または直線的命令シーケンスか、プロシージャか、あるいはその他の命令群である。本発明の好ましい実施形態では、コード・ブロックは、動的変換ルーチン15によって活用される変換の単位である。この変換の単位は、例えば、プロシージャ、基本ブロック、ページ、キャッシュ線、単一命令、またはその他の命令の組み合わせであることができる。

【0019】ブロック情報マップへのアドレス25の各エントリは、アプリケーション内のコード・ブロックまたは共有ライブラリ内のコード・ブロックの開始アドレスを識別する少くとも1つの第1のプログラム・アドレスを含む。加えて、各エントリは、アプリケーションまたは共有ライブラリからのコード・ブロックに関するブロック情報を含むブロック情報テーブル26における位置をポイントするブロック情報ポインタを含む。ブロック情報テーブル26内のブロック情報は、そのコード・ブロックに関する変換されたブロック・コードが存在する場合変換されたコードに対するポインタを含む。

【0020】ステップ23において、実行されるべき次のアプリケーション・アドレスを調べた後、アプリケーションまたは共有ライブラリからのコード・ブロックに関する変換されたコードが存在するか否かの判断が行われる。これは、例えば、ブロック情報テーブル26から取り出された情報を使用して行われる。そのコード・ブロックに関して変換が存在しなければ、ステップ24においてコード・ブロックが変換される。変換されたコードが次に実行される。

【0021】変換コード・キャッシュ20を使用して、効率的な実行のため変換されたコード・ブロックが保持される。変換コード・キャッシュ20によって保持されるコード・ブロックの数は、例えば使用可能メモリに応じて変わる。図4において、変換コード・キャッシュ20内のコード・ブロックは、コード・ブロック27、コード・ブロック28およびコード・ブロック29によって表されている。これらのコード・ブロックのいずれも共有ライブラリからの変換されたプロシージャを表す場合もある。

【0022】変換コード・キャッシュ20内のコード・ブロックは、変換コード・キャッシュ20内の他のコード・ブロックに分岐することもある。例えば、コード・ブロック27内の分岐命令がコード・ブロック28内の位置をポイントする。同様に、コード・ブロック28のコードを実行した後、プログラムの流れ制御がコード・ブロック29へ続く。図4で示されるように、コード・ブロック27のコードの実行後、プログラムの流れ制御は、変換コード・キャッシュ20の外側のコード・ブロックに続くこともある。同様に、コード・ブロック29のコードの実行後、動的分岐が、変換コード・キャッシュ20の外側のコード・ブロック内のアドレスへ分岐する。

10

20

30

40

50

【表1】ソース

行番号	コード
5	sum=0
6	for (i=0; i<array_size; i++)
7	sum += array[i]; ;行5の目的コード STW %gr0, sum(DP) ;sumに0をストア ;行6の目的コード STW %gr0, i(DP) ;iを0に初期化 LDW i(DP), %tr1 ;一時レジスタにiをロード loop_back LDW array_size(DP), %tr2 ;一時レジスタにarray_sizeをロード COMB, >=, n %tr2, %tr1, exit_loop ;if(i>=array_size) ループを出る ;行7の目的コード ADDIL array(DP), %tr3 ;arrayのアドレスを入手 LDWX, s, m %tr1, (%tr3), %tr4 ;array[i]を入手 LDW sum(DP), %tr5 ;sumを入手 ADD %tr4, %tr5, %tr6 ;sumとarray[i]を加算 STW %tr6, 0(%tr3) ;新しいsumをストア LDO 1(%tr1), %tr1 ;i++ B, n loop_back ;次の反復を実行 exit_loop

次の表2は、表1のソース・コードがコンパイルされた結果の目的コードを例示する。

【0032】

【表2】

【0033】表2の目的コードがプロファイル作成機能を持つ動的変換の下で実行される時、動的変換ルーチン15は、頻繁に実行されるコード部分を識別するためプロファイル作成コードを追加する。これによってコンパイラはより高品質のコードを生成することが可能となる。これは、例えば、カウンタを分岐に関連づけること(弓形カウント)、あるいはカウンタを各行を表すコード

に関連づけること(行頻度カウント)のいずれかによって実施される。

【0034】次の表3は、弓形カウントを実行するように動的変換ルーチン15によって追加されたプロファイル作成コードを組み込まれた表2の目的コードを示す。

【0035】

【表3】

```

;行5の目的コード
STW %gr0, sum(DP) ;sumに0をストア
;行6の目的コード
;プロファイル作成コード(パート1)
LDW Line6Hit(DP), %tr8 ;行6を通過した回数を示すカウンタを入手
ADDI, <>1, %tr8, %tr8 ;カウンタを増分
LDI -1, %tr8 ;オーバーフローの場合カウンタを一杯にする
STW %tr8, Line6Hit(DP) ;カウンタをストア
STW %gr0, i(DP) ;iを0に初期化
LDW i(DP), %tr1 ;一時レジスタにiをロード
loop_back
LDW array_size(DP), %tr2 ;一時レジスタにarray_sizeをロード
COMB, >=, n %tr2, %tr1, exit_loop ;if(i>=array_size) ループを出る
;プロファイル作成コード(パート2)
LDW Line6NotTaken(DP), %tr8 ;行6を通過した回数を示すカウンタを入手
ADDI, <>1, %tr8, %tr8 ;カウンタを増分
LDI -1, %tr8 ;オーバーフローの場合カウンタを一杯にする
STW %tr8, Line6NotTaken(DP) ;カウンタをストア
;行7の目的コード
ADDIL array(DP), %tr3 ;arrayのアドレスを入手
LDWX, s, m %tr1, (%tr3), %tr4 ;array[i]を入手

```

```
;sumを入手
;sumとarray[i]を加算
;新しいsumをストア
;i++
;次の反復を実行
```

ドに現れないが、(シンボル・テーブルまたはデバッグ情報のように)非実行部分のアプリケーション・ファイルに含めることができる。例えば、次の表 4 に示されるように、コンパイラは、行 6 に関する詳細なループ情報を要求する注釈をファイルに記述する。

【表 4】

<Label X>	Loop_Detail:	Iteration_Bound=(array_size-1)
-----------	--------------	--------------------------------

的変換ルーチン 15 は、反復の境界がどこに位置しているか知る必要がある。上述の例では、反復の境界は、変数“array\_size”(アレイの大きさ)に含まれていて、1 を減ずることによって調節する必要がある。

【 0 0 4 0 】

【表 5】

```

;sumに0をストア

il>array_size-1}に関する

;ループ反復限界を入手
;分析プロシージャを呼び出す
;反復限界を調整
;iを0に初期化
;一時レジスタにiをロード

;一時レジスタにarray_sizeをロード
_loop ;if(i>=array_size) ループを出る

;arrayのアドレスを入手
4 ;array[i]を入手
;sumを入手
;sumとarray[i]を加算
;新しいsumをストア
;i++
;次の反復を実行

```

50 される。この情報はコンパイラもユーザも利用することができる。ユーザは“Loop-Detail”のような新しい注釈

を加えることによって、追加情報を要求するか、コンパイラが所望のコードを生成するようにコンパイラにヒントを与えることができる。

【0042】以上は本発明の典型的な実施形態を開示し記述したものにすぎない。本発明をその理念および基本的特徴から逸脱することなく別の形態で実施することが可能である点は当業者に認められるべきものである。

【0043】本発明には、例として次のような実施様態が含まれる。

(1) アプリケーションに関する最適化された目的コードを生成する方法であって、アプリケーションのソース・コードをコンパイルして該アプリケーションに関する第1の目的コードを生成するステップ(a)と、実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを含む第2の目的コードに上記第1の目的コードを変換し(b1)、該第2の目的コードを実行してプロファイル情報を生成する(b2)ことを含む上記第1の目的コードに関するプロファイル情報を生成するステップ(b)と、上記プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成するステップ(c)と、を含む最適目的コード生成方法。

(2) 上記ステップ(a)に続いて、該アプリケーションに関して上記第1の目的コードをテストし、妥当性を検査するステップ(d)を更に含む上記(1)に記載の最適目的コード生成方法。

(3) 上記ステップ(c)に続いて、実行の際に追加プロファイル情報を生成する機能を持つプロファイル作成コードを含む第3の目的コードに上記最適化された目的コードを変換し(d1)、該第3の目的コードを実行して追加プロファイル情報を生成する(d2)ことを含む上記最適化された目的コードに関する追加プロファイル情報を生成するステップ(d)と、上記追加プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、更に最適化された目的コードを生成するステップ(e)と、を含む上記(1)に記載の最適目的コード生成方法。

(4) 上記ステップ(a)が上記第1の目的コードに注釈を挿入することを含む、上記(1)に記載の最適目的コード生成方法。

(5) 上記ステップ(b1)において、第1の目的コード内の注釈が第2の目的コード内のプロファイル作成コードの内容を決定する、上記(4)に記載の最適目的コード生成方法。

(6) 上記ステップ(a)において、ソース・コードをコンパイルするコンパイラによって注釈が第1の目的コードに置かれる、上記(5)に記載の最適目的コード生成方法。

(7) 上記ステップ(a)において、ユーザによって注釈が第1の目的コードに置かれる、上記(5)に記載の最

適目的コード生成方法。

(8) 上記ステップ(c)に続いて、実行の際に追加プロファイル情報を生成する機能を持つプロファイル作成コードを含む第3の目的コードに上記最適化された目的コードを変換し(d1)、該第3の目的コードを実行して追加プロファイル情報を生成する(d2)ことを含む上記最適化された目的コードに関する追加プロファイル情報を生成するステップ(d)と、上記追加プロファイル情報を使用して上記ソース・コードを再コンパイルすることによって、更に最適化された目的コードを生成するステップ(e)と、を含む上記(5)に記載の最適目的コード生成方法。

(9) 上記第1の目的コードおよび上記第2の目的コードが同じ機械命令セットを利用する、上記(1)に記載の最適目的コード生成方法。

【0044】(10) アプリケーションに関する第1の目的コードを第2の目的コードに変換し、実行の際にプロファイル情報を生成する機能を持つプロファイル作成コードを該第2の目的コードに追加する変換手段と、該第2の目的コードを実行してプロファイル情報を生成する手段と、を備えるコンピュータ・システム。

(11) 該アプリケーションに関して上記第1の目的コードをテストし、妥当性を検査する手段を更に備える、上記(10)に記載のコンピュータ・システム。

(12) ソース・コードをコンパイルして上記第1の目的コードを生成するコンパイラを更に備える、上記(10)に記載のコンピュータ・システム。

(13) 上記第2の目的コードの実行によって生成された上記プロファイル情報を利用して上記ソース・コードを再コンパイルすることによって、最適化された目的コードを生成する手段を備える、上記(12)に記載のコンピュータ・システム。

(14) 上記コンパイラが上記第1の目的コードに注釈を挿入する、上記(12)に記載のコンピュータ・システム。

(15) 上記変換手段が上記第1の目的コード内の上記注釈を利用して上記第2の目的コード内の上記プロファイル作成コードの内容を決定する、上記(14)に記載のコンピュータ・システム。

(16) ユーザが上記注釈を第1の目的コードに置く手段を含み、上記変換手段が上記第1の目的コード内の該注釈を利用して上記第2の目的コード内の上記プロファイル作成コードの内容を決定する、上記(12)に記載のコンピュータ・システム。

(17) 上記第2の目的コードのサブセットを持つモジュールを記憶する変換コード・キャッシュを更に備える上記(10)に記載のコンピュータ・システム。

(18) ブロック情報マップへのアドレスと、ブロック情報テーブルと、を更に備える上記(17)に記載のコンピュータ・システム。



(19) 上記第1の目的コードおよび上記第2の目的コードが同じ機械命令セットを利用する、上記(10)に記載のコンピュータ・システム。

#### 【0045】

【発明の効果】本発明によって、プロファイル作成コードを備えた特別版の目的コードを生成するため独立したコンパイルを実行する必要性なしに、プロファイルに基づくプログラムの最適化が可能になる。プロファイル・データを生成するため、本発明に従った動的変換ルーチンを使用することによって、アプリケーション開発者のPBO使用が単純化される。更に、本発明の動的変換システムは、動的データ収集に容易に対応でき、PBOに対する反復的アプローチにおける注釈の使用を可能にする。これによって、一層効率的で大規模な実行時プロファイル・データ収集が可能になる。

#### 【図面の簡単な説明】

【図1】本発明の好ましい実施形態に従う動的変換ルーチンを含むコンピュータ・システムのブロック図である。

【図2】本発明の好ましい実施形態に従って動的変換ルーチンを利用してプロファイル・データを生成するシステムの動作の流れ図である。

【図3】本発明の好ましい実施形態に従ってコードを動的に変換する動的変換ルーチンの使用を示すブロック図である。

【図4】本発明の好ましい実施形態に従う動的変換ルーチンによって使用されるブロック情報マップへのアドレス、ブロック情報テーブルおよび変換されたコード・キャッシュを示すブロック図である。

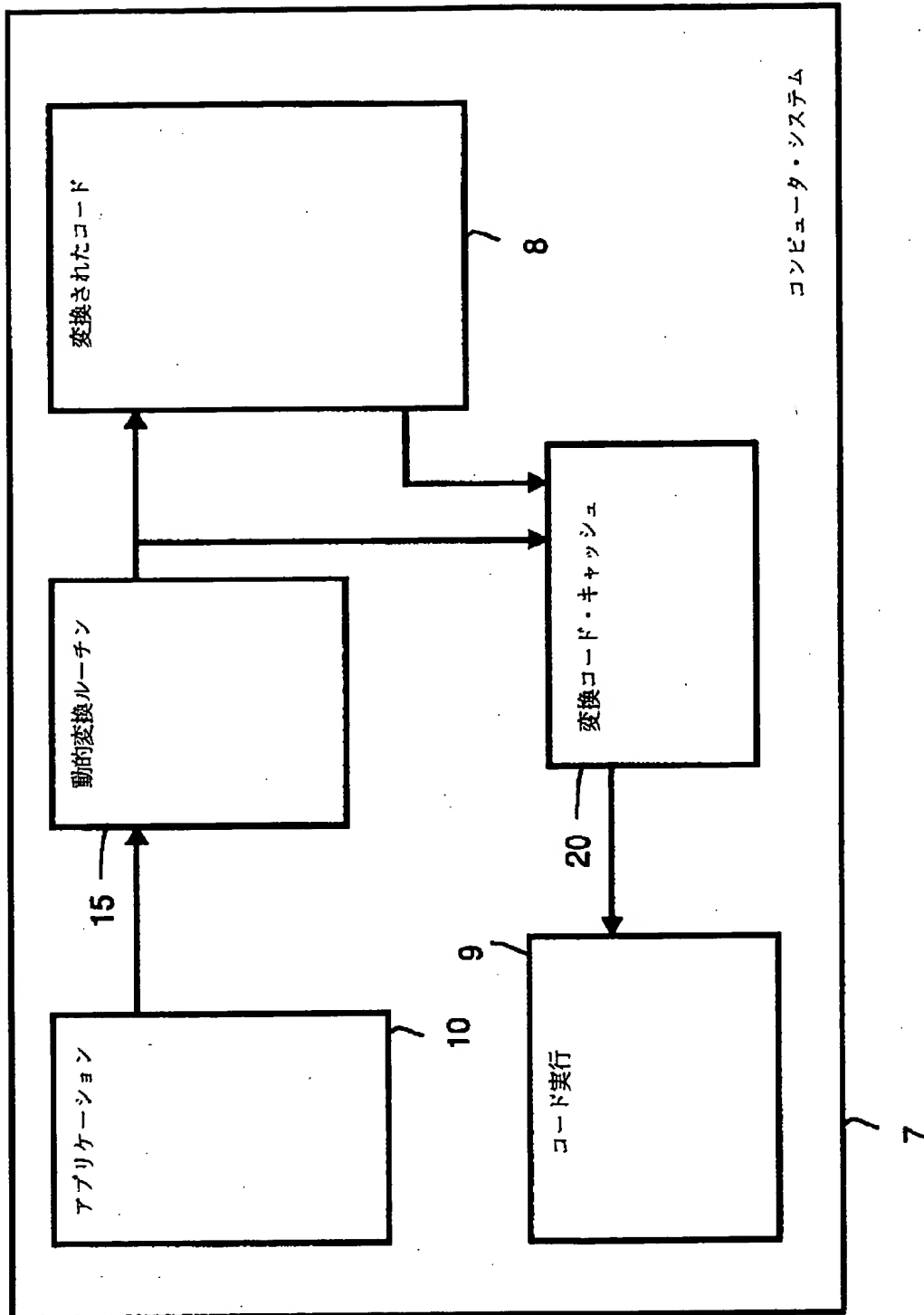
【図5】本発明の好ましい実施形態に従って動的変換ルーチンおよびコード注釈を利用してプロファイル・データを生成するシステムの動作の流れ図である。

【図6】本発明の好ましい実施形態に従って動的変換ルーチンおよびユーザ修正可能コード注釈を利用してプロファイル・データを生成するシステムの動作の流れ図である。

#### 【符号の説明】

- 7 コンピュータ・システム
- 10、33 アプリケーション
- 15、44、54 動的変換ルーチン
- 20 変換コード・キャッシュ
- 25 ブロック情報マップへのアドレス
- 26 ブロック情報テーブル
- 27、28、29 変換目的コード
- 31、41、51 ソース・コード
- 36、45、55 プロファイル情報
- 38、43、53 最適化されたアプリケーション
- 39 動的変換・実行プロセス
- 42、52 コンパイラ
- 56 ユーザ

【図1】

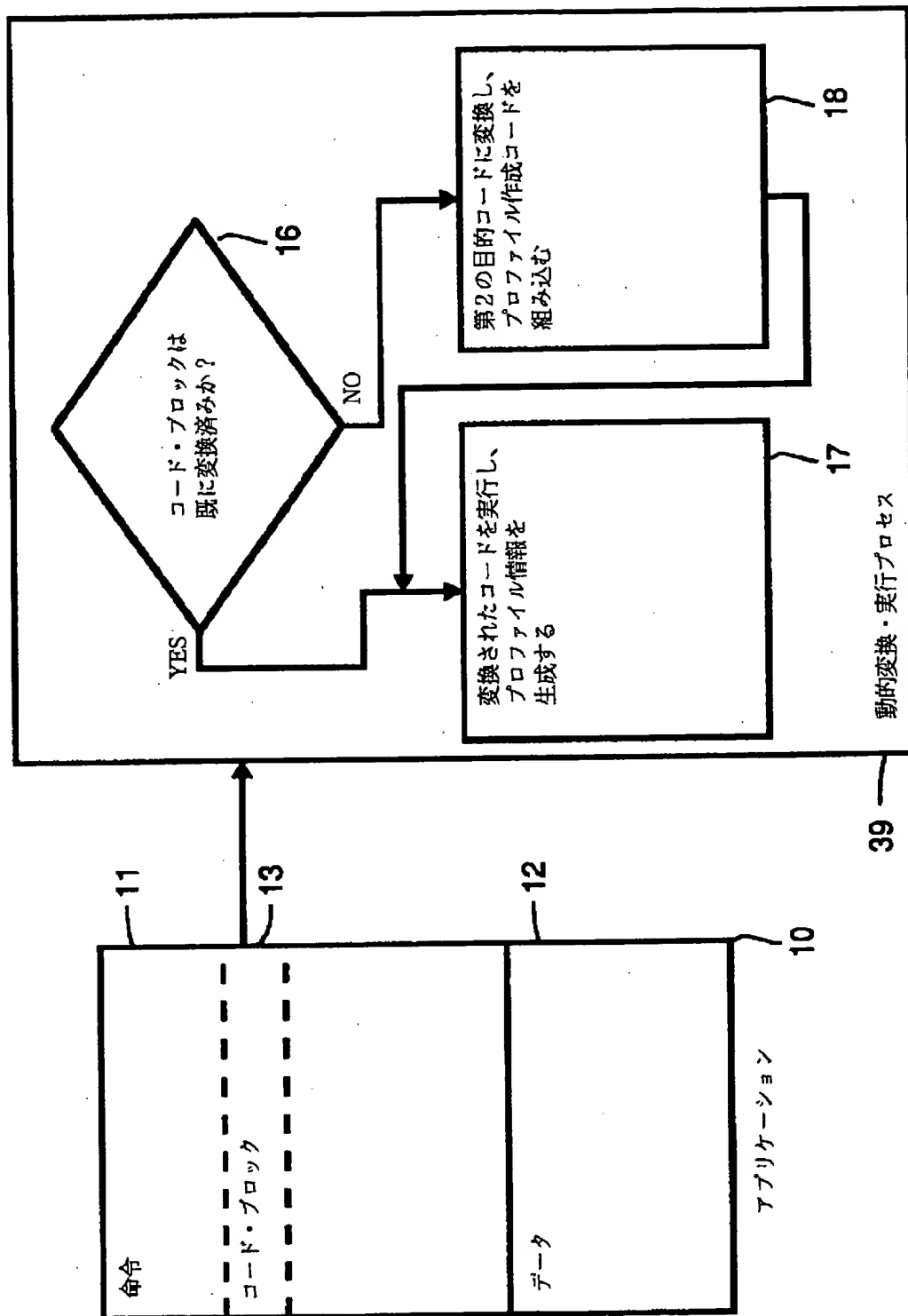


```

graph TD
    31[ソース・コード] --> 32[第1のコンパイル:  
アプリケーションの作成]
    31 --> 37[第2のコンパイル:  
プロファイル情報を用いて  
最適化された  
アプリケーションを生成する]
    32 --> 33((アプリケーション))
    33 --> 34[アプリケーションの  
テストおよび  
妥当性の検査]
    33 --> 35[動的変換ルーチンの下で実行し、  
プロファイル情報を収集する]
    35 --> 36[プロファイル情報]
    36 --> 37
    37 --> 38((最適化された  
アプリケーション))
  
```

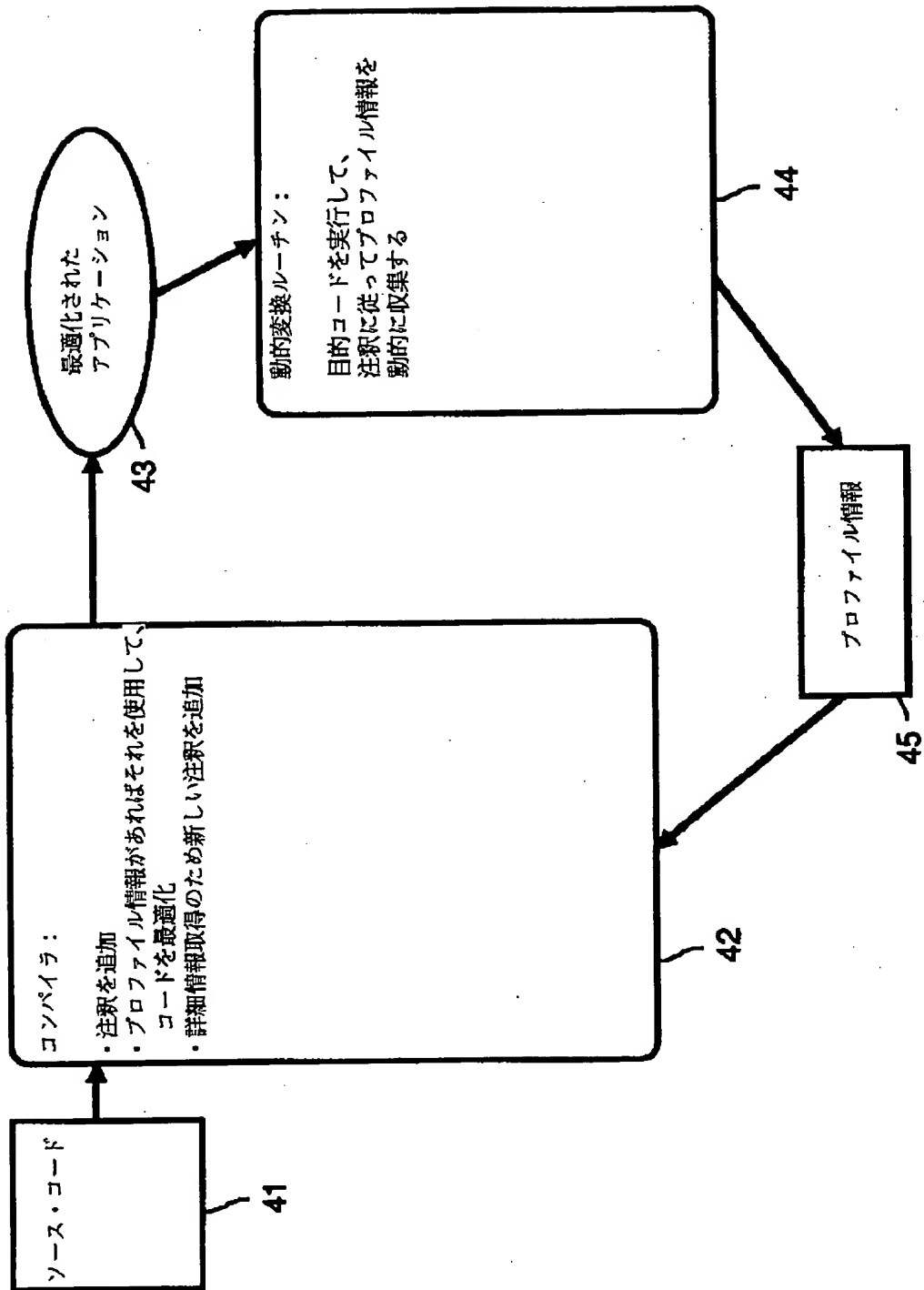
Figure 1 is a flowchart illustrating the process of generating an application. The process begins with a source code (31) which is used to create a first compilation (32) and a second compilation (37). The first compilation (32) leads to an application (33), which is then tested and checked for appropriateness (34). The application (33) is also executed under a dynamic conversion routine to collect profile information (35). This profile information (36) is then used in the second compilation (37) to generate an optimized application (38).

【図3】



[illegible]

【図5】



【図6】

